The Analysis of Convolutional Codes via the Extended Smith Algorithm

R. J. McEliece
California Institute of Technology, Department of Electrical Engineering

I. Onyszchuk
Communications Systems Research Section

Convolutional codes have been the central part of most error-control systems in deep-space communication for many years. Almost all such applications, however, have used the restricted class of (n,1), also known as "rate 1/n," convolutional codes. The more general class of (n,k) convolutional codes contains many potentially useful codes, but their algebraic theory is difficult and has proved to be a stumbling block in the evolution of convolutional coding systems. In this article, the situation is improved by describing a set of practical algorithms for computing certain basic things about a convolutional code (among them the degree, the Forney indices, a minimal generator matrix, and a parity-check matrix), which are usually needed before a system using the code can be built. The approach is based on the classic Forney theory for convolutional codes, together with the extended Smith algorithm for polynomial matrices, which is introduced in this article.

I. Introduction

In his celebrated paper on the algebraic structure of convolutional codes, Forney [2] showed that by using the algebra of $k \times n$ polynomial matrices, in particular the invariant-factor theorem (aka the Smith Form), one can transform an arbitrary generator matrix for an (n, k) convolutional code C into a noncatastrophic, basic, and ultimately minimal, generator matrix for C. He also showed how to find a polynomial inverse for a basic generator matrix for C, and a basic generator matrix for the dual code $C^{\perp}$. In this article, efficient ways are discussed to do all these things. The main tool is an algorithm, called the extended Smith algorithm, which is used to find the invariant factors of an arbitrary $k \times n$ matrix over an Euclidean domain, which bears the same relationship to the usual invariant factor algorithm as the extended Euclid's algorithm bears to the usual Euclid's algorithm.

A brief review of Euclid's algorithm (see e.g. [5, Section 1.1] or [9, Chapter 2]) is helpful. The algorithm's goal is to take a pair (a, b) of elements from a Euclidean domain R, and by repeated application of the division algorithm for R to find the greatest common divisor d of a and b. The goal of the *extended Euclidean algorithm* (see e.g. [6, Section 4.5.2] or [8, Section 8.4]) is to take the same pair, and not only find d, but also find elements s and t of R, such that $sa+tb = d$. The extended Euclidean algorithm has several important applications in coding theory. For example, it can be used to compute inverses in finite fields [9, Exam-

ple 4.2], decode BCH codes [8, Section 8.5], and to find finite impulse response (FIR) inverses for noncatastrophic $(n, 1)$ convolutional generator matrices [1, Section 12.2]. Similarly, the goal of the *Smith algorithm* (see e.g. [3, Section 6.2.4], [4, Section 6.3.3], or Smith's nineteenth-century original article [10, Section 12.2]) is to take an arbitrary $k \times n$ matrix G (with $k \leq n$) over an Euclidean domain R, and by a sequence of elementary row and column operations, to reduce G to a $k \times n$ diagonal matrix $\Gamma = \text{diag}(\gamma_1, \ldots, \gamma_k)$, whose diagonal entries are the invariant factors of G, i.e., $\gamma_i = \Delta_i / \Delta_{i-1}$, where Δ_i is the greatest common denominator of the $i \times i$ minors of G. (Here, $\Delta_0 = 1$ is taken by convention.) Smith's algorithm is reviewed in Section II.

The goal of the *extended Smith algorithm*, which is introduced in this article, is to take the same input, and not only find Γ, but also to find a $k \times k$ unimodular matrix X, and an $n \times n$ unimodular matrix Y, such that $XGY = \Gamma$. It is worthwhile to note that in the special case $k = 1$ and $n = 2$, the (extended) Smith algorithm reduces to the (extended) Euclidean algorithm. The extended Smith algorithm is described in detail in Section III.

Section IV describes how, given an arbitrary generator matrix G for an (n, k) convolutional code, the extended Smith algorithm can be used to efficiently compute the things mentioned above (the degree, the Forney indices, a minimal generator matrix, a polynomial inverse, a parity-check matrix, etc., for the given code). Throughout this article, all results are illustrated with an example of a 2×4 matrix of polynomials over the binary field $GF(2)$, which is a generator matrix for a $(4, 2)$ convolutional code over $GF(2)$.

II. Smith's Algorithm

In this section, a careful description is given of Smith's algorithm (which is often called the invariant-factor algorithm), but a formal proof of its correctness is not given. For that, refer to [3, Section 6.2.4], [4, Section 6.3.3], or [11, Section 12.2].

Begin by recalling the definition of a general Euclidean domain [8, Chapter 2]. It is an integral domain, i.e., a ring that satisfies the cancellation property, together with a "size" function $|a|$ defined for every nonzero element $a \in R$. The size function must satisfy $|a| \leq |ab|$ if $b \neq 0$. Furthermore, if a and b are arbitrary, and $b \neq 0$, a can be "divided" by b in the sense that there exist elements q (the "quotient") and r (the "remainder") such that $a = qb + r$, where either $r = 0$ or else $|r| < |b|$. For application to the study of convolutional codes, always take R to the ring of polynomials over a field F, where the "size" of a polynomial is its degree. However, there are many other Euclidean domains [8, Chapter 2], and Smith's algorithm applies equally to all of them.

The central part of Smith's algorithm is the following subalgorithm E, which takes as input an arbitrary matrix A with entries from an Euclidean domain R, at least one of which is nonzero, and, via elementary row and column operations, transforms A into a matrix with the $(1, 1)$ entry nonzero, every other entry in row 1 and column 1 zero, and every other entry in the matrix divisible by the $(1, 1)$ entry.

E1. Move the entry in A of least size to position $(1, 1)$.

E2. If there is a nonzero entry in either row 1 or column 1 that is not divisible by a_{11}, use it to reduce the size of a_{11}, as follows:

E2a. If a_{1j}, the entry in row 1 and column j, is not divisible by a_{11}, then by the division algorithm there exist nonzero elements q and r such that $a_{1j} = qa_{11} + r$, with $|r| < |a_{1,1}|$. Thus, if q times column 1 is subtracted from column j, the $(1, j)$ entry will be changed to r, which has a smaller size than $a_{1,1}$. If this entry is moved to position $(1, 1)$, the size of a_{11} will have been reduced.

E2b. If a_{i1}, the entry in row i and column 1, is not divisible by a_{11}, repeat the procedure outlined in step E2a, with rows and columns interchanged.

E3. Reduce all the entries in row 1 and column 1 (except $a_{1,1}$ itself) to zero, as follows:

E3a. Since (by step E2) a_{1j} for $j \geq 2$ is divisible by a_{11}, $a_{1j} = q_j a_{11}$. Thus, by subtracting q_j times column 1 from column j, the $(1, j)$ entry will be transformed to zero. Repeating this step for $j = 2, \ldots, r$, the first row will "zero out."

E3b. Repeat step E3a with rows and columns interchanged.

E4. If there is a nonzero entry in A, say a_{ij}, which is not divisible by a_{11}, add column j to column 1. (This produces a nonzero entry in column 1 that is not divisible by a_{11}.) Return to step E2.

E5. Stop. The matrix A now has the desired property.

In the Smith algorithm itself, one applies subalgorithm E successively to the original matrix G, then to the matrix obtained from G by deleting row 1 and column 1, then by deleting row 2 and column 2, etc., until either an all-zero matrix is encountered, or until all rows and columns have been processed. The result is a $k \times n$ diagonal matrix $\Gamma = \mathrm{diag}(\gamma_1, \ldots, \gamma_k)$, whose diagonal entries are the invariant factors of G. The following example illustrates the Smith algorithm, when the underlying Euclidean domain is the ring of polynomials over the field $GF(2)$, in which the "size" of an element is its degree.

Example 1. Consider the following 2×4 matrix with entries that are polynomials over $GF(2)$:

$$G = G_0 = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}$$

Since the lowest degree term already appears in the $(1,1)$ position, skip step E1. Since the $(1,1)$ entry is 1, every nonzero entry in row 1 and column 1 is divisible by the $(1,1)$ entry, so skip step E2 as well. Executing step E3: "zero out" the first row by adding $1+D+D^2$ times column 1 to column 2, $1+D^2$ times column 1 to column 3, and $1+D$ times column 1 to column 4, thereby obtaining successively

$$G_1 = \begin{pmatrix} 1 & 0 & 1+D^2 & 1+D \\ D & 1+D^3 & D^2 & 1 \end{pmatrix}$$

$$G_2 = \begin{pmatrix} 1 & 0 & 0 & 1+D \\ D & 1+D^3 & D+D^2+D^3 & 1 \end{pmatrix}$$

$$G_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1+D+D^2 \end{pmatrix}$$

To zero the entry in position $(2,1)$, add D times row 1 to row 2, thereby obtaining

$$G_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D^3 & D+D^2+D^3 & 1+D+D^2 \end{pmatrix}$$

This completes the operation of subalgorithm E on the original matrix G. Now apply subalgorithm E to the 1×3 matrix obtained by deleting row 1 and column 1 from G_4. The lowest degree term in G_4 appears in position $(2,4)$, so by interchanging columns 2 and 4, it is moved to position $(2,2)$:

$$G_5 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & D+D^2+D^3 & 1+D^3 \end{pmatrix}$$

At this point, all of the remaining entries in row 2 are divisible by the $(2,2)$ entry: $D+D^2+D^3 = D\cdot(1+D+D^2)$, and $1+D^3 = (1+D)\cdot(1+D+D^2)$. Thus by adding D times column 2 to column 3, and then $1+D$ times column 2 to column 4, compute successively

$$G_6 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 1+D^3 \end{pmatrix}$$

$$G_7 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{pmatrix}$$

This completes the operation of the Smith algorithm on G. The matrix G_7 is the invariant-factor form for G; in particular, the invariant factors of G are $\gamma_1 = 1$ and $\gamma_2 = 1+D+D^2$. ∎

III. The Extended Smith Algorithm

The previous section showed, in computational detail, how the Smith algorithm works. Beginning with the matrix $G_0 = G$, it produces a sequence of $k \times n$ matrices G_i, where G_{i+1} is derived from G_i by either an elementary row operation or an elementary column operation. This is represented algebraically as

$$G_{i+1} = E_{i+1} G_i F_{i+1} \tag{1}$$

where E_{i+1} and F_{i+1} are $k \times k$ and $n \times n$ elementary matrices, respectively. If G_{i+1} is obtained from G_i via a *row* operation, then E_{i+1} is a nontrivial $k \times k$ elementary matrix, but $F_{i+1} = I_n$. If G_{i+1} is obtained from G_i via a *column* operation, then F_{i+1} is a nontrivial $n \times n$ elementary matrix, but $E_{i+1} = I_k$. After a finite number N of steps, $G_N = \Gamma$.

The extended Smith algorithm builds on the Smith algorithm. Whereas the Smith algorithm works only with the sequence $G_0, G_1, \ldots, G_N$, the extended Smith algorithm also works with a sequence of unimodular $k \times k$ matrices $X_0, \ldots, X_N$, and a sequence of unimodular $n \times n$ matrices $Y_0, \ldots, Y_N$. The sequences (X_i) and (Y_i) are initialized as $X_0 = I_k$ and $Y_0 = I_n$, and updated via the rule [cf. Eq. (1)]

$$X_{i+1} = E_{i+1} X_i \tag{2}$$

$$Y_{i+1} = Y_i F_{i+1} \tag{3}$$

The following simple lemma is the key to the extended Smith algorithm.

Lemma.

$$X_i G Y_i = G_i \qquad \text{for } i = 0, 1, \ldots, N \tag{4}$$

Proof. Since $X_0 = I_k$ and $Y_0 = I_n$, Eq. (4) holds for $i = 0$. Assuming now that Eq. (4) holds for a given value of i, multiply both sides of Eq. (4) on the left by E_{i+1} and on the right by F_{i+1}

$$E_{i+1} X_i G Y_i F_{i+1} = E_{i+1} G_i F_{i+1} \tag{5}$$

By Eq. (1), the right side of Eq. (5) is equal to G_{i+1}. Thus

$$(E_{i+1} X_i) G (Y_i F_{i+1}) = G_{i+1} \tag{6}$$

But $E_{i+1} X_i = X_{i+1}$ by Eq. (2) and $Y_i F_{i+1} = Y_{i+1}$ by Eq. (3), so Eq. (6) becomes simply $X_{i+1} G Y_{i+1} = G_{i+1}$, which proves Eq. (4) by induction. ∎

If Eq. (4) is specialized with $i = N$,

$$X_N G Y_N = \Gamma \tag{7}$$

which is the desired extended Smith diagonalization of G. A convenient way to implement the extended Smith algorithm is to extend the original matrix G to dimensions $(n + k) \times (n + k)$ as follows:

$$G' = \begin{pmatrix} G & I_k \\ I_n & 0_{n \times k} \end{pmatrix} \tag{8}$$

where in Eq. (8) $0_{n \times k}$ is an $n \times k$ matrix of zeros. Then if the sequence of elementary row and column operations generated by the Smith algorithm applied to G is performed on the extended matrix G', after i iterations, the resulting matrix G'_i has the form

$$G'_i = \begin{pmatrix} G_i & X_i \\ Y_i & 0_{n \times k} \end{pmatrix} \tag{9}$$

Thus, after N steps, the matrices X_N and Y_N in Eq. (7) appear as the upper-right and lower-left blocks of G'_N, respectively. All this is illustrated by extending the example from Section II.

Example 2. Begin as in Example 1, with the following 2×4 matrix over $GF(2)[D]$:

$$G = G_0 = \begin{pmatrix} 1 & 1 + D + D^2 & 1 + D^2 & 1 + D \\ D & 1 + D + D^2 & D^2 & 1 \end{pmatrix}$$

Then the corresponding matrix G' [cf. Eq. (8)] is

$$G' = G'_0 = \begin{pmatrix} 1 & 1 + D + D^2 & 1 + D^2 & 1 + D & 1 & 0 \\ D & 1 + D + D^2 & D^2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

In Example 1, G_1, G_2, and G_3 are obtained from G_0 by successively adding $1 + D + D^2$ times column 1 to column 2, $1 + D^2$ times column 1 to column 3, and $1 + D$ times column 1 to column 4. If the same sequence of operations is performed to G'_0, one obtains successively

$$G_1' = \begin{pmatrix} 1 & 0 & 1+D^2 & 1+D & 1 & 0 \\ D & 1+D^3 & D^2 & 1 & 0 & 1 \\ 1 & 1+D+D^2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$G_2' = \begin{pmatrix} 1 & 0 & 0 & 1+D & 1 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1 & 0 & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$G_3' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1+D+D^2 & 0 & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 1+D & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Next, adding D times row 1 to row 2, as was done to obtain G_4 from G_3, obtain

$$G_4' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D^3 & D+D^2+D^3 & 1+D+D^2 & D & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 1+D & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Interchanging columns 2 and 4, obtain

$$G_5' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & D+D^2+D^3 & 1+D^3 & D & 1 \\ 1 & 1+D & 1+D^2 & 1+D+D^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Finally, adding D times column 2 to column 3, and $1+D$ times column 2 to column 4, one computes successively

$$G_6' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & 0 & 1+D^3 & D & 1 \\ 1 & 1+D & 1+D & 1+D+D^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & D & 0 & 0 & 0 \end{pmatrix}$$

and

$$G_7' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & 0 & 0 & D & 1 \\ 1 & 1+D & 1+D & D & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & D & 1+D & 0 & 0 \end{pmatrix}$$

Thus the extended Smith decomposition of the original matrix G is given by [cf. Eq. (7)]

$$\begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix} \cdot G \cdot \begin{pmatrix} 1 & 1+D & 1+D & D \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & D & 1+D \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{pmatrix}$$

In Section V, the decomposition of Example 2 will be used in the analysis of the (4, 2) convolutional code generated by G, as an illustration of the general results to be expounded upon in Section IV.

IV. Application to the Analysis of Convolutional Codes

In this section, given an arbitrary generator matrix G for an (n, k) convolutional code C, one can efficiently find, among other things: a basic generator matrix, say G_{basic} for C; a polynomial inverse for G_{basic}; a minimal generator matrix for C; and a basic generator matrix for the dual code $C^{\perp}$. The central tool needed to do all this is the extended Smith algorithm introduced in Section III.

Assume then that G is a $k \times n$ generator matrix for an (n, k) convolutional code over the field F, i.e., a $k \times n$ matrix of rank k over the field $F(D)$ of rational functions over F. By multiplying the ith row of G by the least common multiple of the denominators in that row, one easily obtains an equivalent generator matrix, all of whose entries are polynomials over F, i.e., elements of $F[D]$. Since $F[D]$ is an Euclidean domain, one may apply the extended Smith algorithm described in Section III, thereby obtaining a decomposition of Eq. (7). In what follows, the matrices X_N and Y_N produced by the extended Smith algorithm will be denoted simply by X and Y.

The matrices X, Y, and Γ contain much valuable information about the code C and the generator matrix G. To extract this information, however, first define several useful pieces of these matrices, called Γ_k, $\tilde{\Gamma}_k$, K, and H (in what follows, $r = n - k$)

$$\Gamma_k = \text{leftmost } k \text{ columns of } \Gamma = \text{diag}(\gamma_1, \ldots, \gamma_k)$$

$$\text{(dimensions: } k \times k) \tag{10}$$

$$\tilde{\Gamma}_k = \gamma_k \cdot \Gamma_k^{-1} = \text{diag}(\gamma_k/\gamma_1, \ldots, \gamma_k/\gamma_k)$$

$$\text{(dimensions: } k \times k) \tag{11}$$

$$K = \text{leftmost } k \text{ columns of } Y$$

$$\text{(dimensions: } n \times k) \tag{12}$$

$$H = \text{rightmost } r \text{ columns of } Y$$

$$\text{(dimensions: } n \times r) \tag{13}$$

The following theorem describes the useful outputs of the extended Smith algorithm when applied to G. (For terminology not fully explained here, refer to Forney [2].)

Theorem 1. With the matrices Γ_k, $\tilde{\Gamma}_k$, K, and H defined as in Eqs. (10)–(13), one has the following:

(a) *A basic generator matrix for C:* $G_{\text{basic}} = \Gamma_k^{-1} X G$. (That is, G_{basic} is obtained by dividing the ith row of XG by the invariant factor γ_i, for $i = 1, \ldots, k$.)

(b) *A polynomial inverse for G_{basic}:* K.

(c) *A polynomial pseudo-inverse for G, with factor γ_k:* $K\tilde{\Gamma}_k X$. (In particular, if G is already basic, i.e., if $\Gamma_k = I_k$, then KX is a polynomial inverse for G.)

(d) *A basic generator matrix for $C^{\perp}$, i.e., a parity-check matrix for C:* H^T.

Proof. From Eq. (7) with $X = X_N$ and $Y = Y_N$, one has

$$XG = \Gamma B \tag{14}$$

where $B = Y^{-1}$. Now B is an $n \times n$ unimodular polynomial matrix. Let U be the $k \times n$ matrix consisting of the first k rows of B, and L be the $r \times n$ matrix consisting of the last r rows of B. Then, since $\Gamma = (\Gamma_k \quad 0_{k\times r})$ where $0_{k\times r}$ is a $k \times r$ all-zeros matrix, it follows that

$$\Gamma B = (\Gamma_k \quad 0_{k\times r}) \begin{pmatrix} U \\ L \end{pmatrix} = \Gamma_k U \tag{15}$$

so that, combining Eqs. (14) and (15)

$$U = \Gamma_k^{-1} XG \tag{16}$$

From Eq. (16), U is row equivalent to G. Furthermore, since $U = (I_k \quad 0_{k\times r})B$, and B is unimodular, it follows that the invariant factors of U are all equal to 1, and so U is a basic generator matrix for C. This proves Theorem 1(a).

To prove Theorem 1(b), use the fact that Y and B are inverse matrices, so that

$$BY = \begin{pmatrix} U \\ L \end{pmatrix} (K \quad H) = \begin{pmatrix} UK & UH \\ LK & LH \end{pmatrix}$$

$$= I_n = \begin{pmatrix} I_k & 0_{k\times r} \\ 0_{r\times k} & I_r \end{pmatrix} \tag{17}$$

It follows from Eq. (17) that $UK = I_k$, so that K is a polynomial inverse for the basic generator matrix U, which proves Theorem 1(b).

To prove Theorem 1(c), suppose that J is an $n \times k$ polynomial pseudo-inverse for G with (polynomial) factor $\psi(D)$, i.e., that $GJ = \psi I_k$. Then

$$XGYY^{-1}JX^{-1} = XGJX^{-1} = X(\psi I_k)X^{-1} = \psi I_k \tag{18}$$

But from Eq. (7), $XGY = \Gamma = (\Gamma_k \quad 0_{k\times r})$. Thus, if the matrix $Y^{-1}JX^{-1}$ is denoted by J', J' is a polynomial matrix (since X and Y are unimodular) and Eq. (18) becomes

$$(\Gamma_k \quad 0)\, J' = \psi I_k \tag{19}$$

If now $J' = \begin{pmatrix} J'_0 \\ J'_1 \end{pmatrix}$ where J'_0 is $k \times k$ and J'_1 is $r \times k$, it follows from Eq. (19) that $J'_0 = \psi\Gamma_k^{-1}$. But since J' is a polynomial matrix, and $\Gamma_k^{-1} = \mathrm{diag}(\gamma_1^{-1}, \ldots, \gamma_k^{-1})$, it follows that ψ must be a multiple of γ_k. Conversely, if $\psi = \gamma_k$, so that $J'_0 = \gamma_k\Gamma_k^{-1} = \Gamma_k$ and $J'_1 = 0_{r\times k}$, then $J' = \begin{pmatrix} \tilde{\Gamma}_k \\ 0_{r\times k} \end{pmatrix}$. Thus, $J = YJ'X = K\tilde{\Gamma}_k X$ is a polynomial pseudo-inverse with factor γ_k, which proves Theorem 1(c).

To prove Theorem 1(d), one has to show that the set of codewords in the code generated by G, i.e., the set of n-dimensional $F(D)$-vectors y of the form $y = xG$, is identical to the set of vectors y such that $yH = 0$. Thus, let $Y_1 = \{y : y = xG\}$ and $Y_2 = \{y : yH = 0\}$. It will be shown that $Y_1 = Y_2$.

Since, as in Theorem 1(a), the matrix U, defined to be the first k rows of $B = Y^{-1}$, is (a basic generator matrix) equivalent to G, it follows that $Y_1 = \{y : y = xU\}$. Thus, if $y \in Y_1$, $y = xU$ for some x and so $yH = (xU)H = x(UH) = 0$, since by Eq. (17), $UH = 0_{k\times r}$. Thus, $Y_1 \subseteq Y_2$. On the other hand, if y is arbitrary, then since $B^{-1} = Y$,

$$y = yI_n = y(YB) = y\left((K \quad H)\begin{pmatrix} U \\ L \end{pmatrix}\right)$$

$$= y(KU + HL) = (yK)U + (yH)L \tag{20}$$

Now suppose $y \in Y_2$, i.e., $yH = 0$. Then from Eq. (20) $y = (yK)U = xU$, where $x = yK$, and so $y \in Y_1$. Thus, $Y_1 \subseteq Y_2$, which completes the proof of Theorem 1(d). ∎

In the next section these results will be briefly illustrated with a simple example.

V. An Example Convolutional Code

To illustrate the results of this article, consider the following generator matrix G for a binary $(4, 2)$ convolutional code:

$$G = \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{pmatrix}$$

In Section III the extended invariant-factor decomposition of G was found to be $XGY = \Gamma$, where

$$\Gamma = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{pmatrix}$$

$$X = \begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix}$$

$$Y = \begin{pmatrix} 1 & 1+D & 1+D & D \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & D & 1+D \end{pmatrix}$$

Thus, the matrices defined in Eqs. (10)–(13) are as follows:

$$\Gamma_k = \begin{pmatrix} 1 & 0 \\ 0 & 1+D+D^2 \end{pmatrix}$$

$$\tilde{\Gamma}_k = \begin{pmatrix} 1+D+D^2 & 0 \\ 0 & 1 \end{pmatrix}$$

$$K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1+D & 0 & 0 & 1 \end{pmatrix}^T$$

$$H = \begin{pmatrix} 1+D & 0 & 1 & D \\ D & 1 & 0 & 1+D \end{pmatrix}^T$$

Using the prescriptions in Theorem 1, one quickly obtains the following:

(a) *A basic generator matrix for C:*

$$G_{\text{basic}} = \Gamma_k^{-1} X G$$

$$= \begin{pmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 0 & 1+D & D & 1 \end{pmatrix}$$

(b) *A polynomial inverse for G_{basic}:*

$$K = \begin{pmatrix} 1 & 1+D \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

(c) *A polynomial pseudo-inverse for G, with factor $\gamma_2 = 1+D+D^2$:*

$$K\tilde{\Gamma}_k X = \begin{pmatrix} 1 & 1+D \\ 0 & 0 \\ 0 & 0 \\ D & 1 \end{pmatrix}$$

(d) *A (basic) generator matrix for $C^\perp$:*

$$H^T = \begin{pmatrix} 1+D & 0 & 1 & D \\ D & 1 & 0 & 1+D \end{pmatrix}$$

Finally, note that in this case neither of the basic generator matrices G_{basic} or H^T is minimal. To minimize them, follow the simple algorithm originally described in [2], or perhaps more lucidly in Kailath [4, Section 6.3.2] (where minimal matrices are, however, called row-reduced). The idea is to use elementary row operations on G to reduce the degree of the highest degree terms in some row, as long as this is possible. For example, to minimize G_{basic}, multiply the second row by D and add it to the first row, obtaining

$$G' = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1+D & D & 1 \end{pmatrix}$$

Since elementary row operations cannot reduce the row degrees further, the matrix G' is a minimal generator matrix for the code C. (Indeed, at this point, one can recognize C as the $d_{\text{free}} = 4$ partial-unit-memory $(4,2)$ code first found by Lauer [7, Table 1].) To find a polynomial inverse for G', note that $G' = TG$, where T is a unimodular $k \times k$ polynomial matrix, and so $K' = KT^{-1}$ is a polynomial inverse for G'. In this example, simply multiply the *first column* of K by D and *subtract* it from the *second column*, thereby obtaining the following polynomial inverse for the minimal generator matrix G':

$$K' = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

To minimize H^T, simply add row 2 to row 1, thereby obtaining the following minimal generator matrix for the dual code:

$$H' = \begin{pmatrix} 1 & 1 & 1 & 1 \\ D & 1 & 0 & 1+D \end{pmatrix}$$

In this case, the dual code is isomorphic to the original code.

The *Forney indices* of an (n, k) convolutional code are the degrees of the rows of a minimal generator matrix, and the *degree* of the code is the sum of the Forney indices. In this example, the *degree* of both C and $C^{\perp}$ is 1, and the Forney index of both codes is $(0, 1)$. It is a general theorem that $\deg C = \deg C^{\perp}$ (see [2, Theorem 7]), but the equality of the Forney indices in this case is more or less accidental.

References

[1] R. E. Blahut, *Theory and Practice of Error-Correcting Codes*, Reading, Massachusetts: Addison-Wesley, 1983.

[2] G. D. Forney, "Convolutional Codes I: Algebraic Structure," *IEEE Transactions on Inform. Theory*, vol. IT-16, pp. 720–738, November 1970.

[3] F. R. Gantmacher, *The Theory of Matrices, Volume I*, New York: Chelsea Publishing Company, 1977.

[4] T. Kailath, *Linear Systems*, Englewood Cliffs, N. J.: Prentice-Hall, 1980.

[5] D. E. Knuth, *The Art of Computer Algorithms, Volume 1: Fundamental Algorithms*, second edition, Reading, Massachusetts: Addison-Wesley, 1973.

[6] D. E. Knuth, *The Art of Computer Algorithms, Volume 2: Seminumerical Algorithms*, second edition, Reading, Massachusetts: Addison-Wesley, 1981.

[7] G. S. Lauer, "Some Optimal Partial-Unit-Memory Codes," *IEEE Transactions on Inform. Theory*, vol. IT-25, pp. 240–243, March 1979.

[8] R. J. McEliece, *The Theory of Information and Coding*, Reading, Massachusetts: Addison-Wesley, 1977.

[9] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Boston: Kluwer, 1987.

[10] H. J. S. Smith, "On Systems of Linear Indeterminate Equations and Congruences," *Philos. Transactions of the Royal Society of London*, vol. 151, pp. 293–326, 1861.

[11] B. L. van der Waerden, *Algebra, Volume 2*, New York: Frederick Ungar, 1970.